# MULTI-STREAM MERGE NETWORK FOR DATA WIDTH CONVERSION AND MULTIPLEXING

## FIELD OF THE INVENTION

[0001] This invention relates to the field of data transmission, and in particular to a system and method for converting data between various widths or formats.

## BACKGROUND TO THE INVENTION

[0002] It is frequently necessary to convert data between various widths or formats. For example, interconnecting buses of different widths (e.g. an 8-bit bus to be interfaced with a 32-bit bus) requires that data be transformed from or to an 8-bit format to or from a 32-bit format. As an example of a more specific requirement, a physical layer device for a telecommunications network interface may need to convert data widths between an optical fiber link on one side of the interface and a local system bus on the other. The data received from the optical fiber may consist of 8-bit bytes, but must be placed on a 32-bit system bus as contiguous 32-bit words.

[0003] It is also sometimes necessary to merge different data streams received from separate physical entities into a single stream, or to split one stream among multiple receivers. In this case, the single data stream of high bandwidth would normally be time-division-multiplexed among the several lower-bandwidth streams. For example, 32-bit words carried on a 32-bit bus may have to be split into four streams, with successive 32-bit words being sent to different 8-bit destinations (i.e. the first 32-bit word would be assigned to destination 0, the second to destination 1, etc.). The reverse may also be required, wherein data arriving on four separate 8-bit channels must be accumulated and time-multiplexed to form a single 32-bit

channel. This type of processing finds application wherein a single wide data bus must be interfaced to several narrow data buses, or where several physical layer interface devices must be interfaced to a single wide high-speed local system bus.

**[0004]** Figure 1 illustrates the multiplexing and demultiplexing process. An example of four independent 8-bit data streams A, B, C and D, formed of successive 8-bit words A0, A1, A2…, B0, B1, B2…, C0, C1, C2… and D0, D1, D2…, are input to a merge apparatus 1 which accumulates four bytes at a time from each stream and then outputs parallel 32-bit words to a 32-bit bus 2. Consecutive words 3A, 3B, 3C and 3D carried by the 32-bit bus each consists of four bytes from successive input streams (4 bytes x 8 bits/word = 32 bit words).

**[0005]** Further, a single 32-bit data stream carrying 32-bit words belonging to logically separate channels is sometimes required to be de-multiplexed into four physically separate 8-bit streams.

**[0006]** It is sometimes required for some applications to be able to merge data from, or split data to, different physical data streams of variable widths, while maintaining a wider constant data bus width. For example, a 128-bit bus may be required to be interfaced with a combination of 8-bit and 32-bit buses, with the aggregate bandwidth of the 128-bit bus being split among the narrower buses in direct proportion to their width. This could be required if several physical layer interface devices are to be connected to the same logical system bus, but the interface devices have different data rates and produce data of different widths.

DESCRIPTION OF THE PRIOR ART

[0007] A number of different ways to solve the above-described data width conversion and merging problems have been employed in widely disparate applications, such as those described in the following U.S. patents: 5,016,011 issued May 14, 1991, invented by R.I.Hartley et al; 4,942,396 issued July 17, 1990, invented by R.I. Hartley et al; 5,802,399 issued September 1, 1998,invented by M. Yumoto et al; 4,747,070 issued May 24, 1988, invented by R.R. Trottier et al; 4,309,754 issued January 5, 1982, invented by J.M. Dinwiddie, Jr.; 4,271,480 issued June 2, 1981, invented by D. Vinot; 4,162,534 issued July 24, 1979, invented by G.H. Barnes; 3,800,289 issued March 26, 1974, invented by K.E. Batcher; 3,686,640 issued August 22, 1972, invented by S.R. Andersen et al; 3,934,132 issued January 20, 1976, invented by D.J. Desmonds; 3,747,070 issued July 17, 1973, invented by J.H. Huttenhoff; and 3,812,467 issued May 21, 1974, invented by K.E. Batcher.

[0008] For example, one way in the prior art of solving the merge problem is to use an arrangement of shift registers and multiplexers such that the narrower-width data are shifted into independent, wide, shift registers, one per input data stream, and then the contents of the shift registers are successively multiplexed on to a single wide output bus. The 8-bit to 32-bit conversion example above requires four 32-bit shift registers into which the data streams A, B, C, D are shifted, 8-bits at a time. When a complete 32-bit word becomes available from a particular stream, it is output as a unit on the 32-bit output bus.

[0009] This solution suffers from the problem that the complexity of the logic and routing grows as the square of the size of the output bus; an output bus of 256 bits

coupled to 32 8-bit input buses requires 32 256-bit shift registers and a 32:1 multiplexer that is also 256 bits wide, plus the connection routing area occupied by 32 256-bit data buses. The result is a very large and expensive circuit that is not capable of running at high speeds. In addition, the control complexity becomes substantial when data of different widths must be combined on to the output bus.

**[0010]** Some of the difficulties encountered with the above approach can be solved by utilizing a data random access memory (RAM) to buffer the data. Some degree of reduction in the routing and logic impact may be obtained in this manner. The RAM is required to be operated at a high data rate, high enough to permit data to be written to it from each narrow stream in succession, the narrow streams being multiplexed together on a shared, high-speed data bus. When sufficient data becomes available within the RAM buffer for any one input stream to form a complete word on the wider data bus, the data are read out on to the output bus.

**[0011]** This solution, however, requires the use of a RAM and surrounding logic of extremely high speed, operating at N times the data rate of any one input stream, wherein N is the number of separate streams. This is not feasible or inexpensive when high data rates must be encountered.

**[0012]** Similar structures using individual registers in place of the RAM have also been proposed, but also possess similar problems.

**[0013]** Approaches using shifting networks have been implemented. These are relatively more flexible than the simple shift register mechanism described above, and involve the use of multi-stage shifting networks to shift and align incoming data from narrower streams to various positions in a wider stream, followed by register and buffer logic to

merge the various narrow data words together into the desired time-multiplexed output.

[0014] However, they suffer from the same problems as noted earlier with respect to the shift register approach, in which the complexity of the logic and routing grows as the square of the size of the output bus, and in addition are not feasible at high speeds and/or large data widths.

SUMMARY OF THE PRESENT INVENTION

[0015] An embodiment of the present invention as will be described herein can provide much less complexity of the implementation logic, layout and routing of interconnections than that required by the earlier shift register implementation described earlier. It is simple and regular, and requires only simple control apparatus.

[0016] It can accommodate a number of input data streams, each of which may be a different number of bits in width, and can concatenate consecutive data units from each input stream to produce wider data words which are placed on a wide output data bus.

[0017] The number of individual (atomic) data-units in the output data stream is restricted to a power of 2, and the various widths of the input data streams are restricted to power of 2 multiples of each other. The description and claims herein should be construed to contain this limitation if not otherwise stated. For example, a combination of an output data stream of 128 bits in width and various numbers of input data streams that are 8, 16, 32, 64 and 128 bits wide meets this requirement.

[0018] The invention can be constructed so that data can flow in either direction of merging/demultiplexing, narrow streams being merged into a wide stream, and wide streams being split into several narrow streams.

[0019] The physical assignment of signal lines to channels (i.e. narrow streams) can be arbitrarily modifiable within the power of 2 constraints noted above. For example, considering the example given above, the narrow streams may be distributed in some arbitrary fashion across a set of 128 input signal lines. Thus there may be one 8-bit stream assigned to the first 8 lines, two 32-bit streams assigned to the next 64 lines, three 8-bit streams assigned to the next 32 lines, and one 32 bit stream assigned to the remaining 32 lines.

[0020] In accordance with an embodiment of the invention, a merging network for multiple data streams comprises a pipelined butterfly network, wherein the pipelined butterfly network comprises: (a) an input network for receiving a plurality of data streams of mutually constant widths, each data stream having logically related data bits carried on contiguous signal lines, (b) a butterfly network containing suitably interconnected register and multiplexer means for rearranging the received data streams into a time-multiplexed constant-width output data stream, the output data stream having a width equal to or greater than the sum of the widths of the input data streams, and (c) an output network for providing the output data stream interleaved to an output bus.

[0021] In accordance with another embodiment, a multi-data-stream merge network comprises: (a) a plurality of shuffle buffers for receiving plural input data streams, (i) each data stream but one having a width which is the same as all other input data streams, and (ii) at least one stream of data having different data width than other ones of the input streams of data and in which the data widths of the streams of data have a power of 2 relationship; the shuffle

buffer having a structure for reordering the input streams of data into an interleaved order if not already in interleaved order and providing the interleaved order of data streams at its output, (b) a permutation network for receiving the interleaved order of data streams and rearranging the spatial order of the data streams if desirable or necessary and locating each stream on a specific spatial boundary, and (c) a pipelined butterfly network for receiving the data streams from the permutation network and concatenating data from the received data streams into constant width interleaved words having a width which is wider than any of the input streams of data, and merging the constant width words onto an output bus in a time-division-multiplexed manner.

**[0022]** In accordance with another embodiment, a multi-data-stream merge network comprises a pipelined butterfly network for receiving streams of data which are in interleaved order and which are located at specific spatial boundaries, and includes apparatus for concatenating data from the received streams of data into constant width interleaved words having a width which is wider than any of the received streams of data, and apparatus for merging the constant width words onto an output bus in a time-division multiplexed form so as to produce a constant-width interleaved output data stream.

BRIEF INTRODUCTION TO THE DRAWINGS

**[0023]** A better understanding of the invention may be obtained by reading the detailed description of the invention below, in conjunction with the following drawings, in which:

**[0024]** Figure 1 is a block diagram illustrating a multiplexing and demultiplexing process achieved by the

present invention,

[0025] Figure 2 is a block diagram illustrating one embodiment of the invention,

[0026] Figure 3 is a block diagram illustrating another embodiment of the invention,

[0027] Figure 4 is a block diagram of a shuffle buffer used in the group of shuffle buffers described in figures 2 and 3,

[0028] Figure 5 is a block diagram of a Benes network of which the permutation network may be comprised,

[0029] Figure 6 is a block diagram of a node used in the Benes network of figure 5,

[0030] Figure 7 is a block diagram of a pipelined butterfly network,

[0031] Figures 8A - 8F depict the pipelined butterfly network of Figure 7 in successive clock cycles, illustrating passage of data therethrough,

[0032] Figure 9 is a block diagram of a pipelined butterfly network having a larger number of inputs than the pipelined butterfly network of Figure 7, and

[0033] Figure 10 is a block diagram of an embodiment of a system which includes a pipelined butterfly network and input shuffle buffer network for handling data streams of different widths in the same apparatus.

DETAILED DESCRIPTION OF EMBODIMENTS OF THE INVENTION

[0034] Turning to Figure 2, a plurality of input buses 5 carry mixed width data streams, having the power of 2 relationship described above, i.e. the various widths of the input data streams are restricted to power of 2 multiples of each other.

[0035] The mixed width data streams are input to a shuffle buffer network 7 formed of a plurality of shuffle

buffers. The shuffle buffer network accepts, buffers and reorders incoming data from the multiple different-data-width input buses, if required, as will be described later.

[0036] The output of the shuffle buffer system 7 is applied to a permutation network 9. The permutation network 9 rearranges the different data streams to create interleaved groups based on the input data width, again if necessary as will be described later.

[0037] The output of the permutation network 9 is applied to a pipelined butterfly network 11, which outputs its data to a wide output bus 13. The pipelined butterfly network performs the actual merging and data width conversion process on the input data streams to produce a time-division-multiplexed output stream.

[0038] It should be noted that these three elements may be used in reverse order, to accomplish the reverse process of accepting a single wide time-multiplexed data stream, splitting it into its constituent parts, and outputting them as multiple narrower streams of different data widths. Due to this operational reciprocity, keeping in mind that the splitting operation is but a reversal of the merging operation, a detailed description of the splitting operation would be redundant to a person skilled in the art. Therefore while a detailed description of the structure of and operation of the system for performing the merging operation will be given, only a brief description, as given later, of the splitting operation is believed necessary for a person skilled in the art.

[0039] Concerning operation of the system illustrated in Figure 2, a number of constant streams of data 5, possibly of several data widths, are presented to the inputs of the shuffle buffers 7, into which the data is written. The data

stored in the shuffle buffers is read out in accordance with a specific process to be described later, and is presented to the input of the permutation network 9. The permutation network 9 rearranges the streams and applies the rearranged streams to the inputs of the pipelined butterfly network 11. The pipelined butterfly network 11 concatenates data from each input stream separately into wider data words (of constant width, regardless of the width of the input stream) and then merges the words for different streams on to the single output bus in a time-division-multiplexed manner.

**[0040]** The output of the pipelined butterfly network 11 is interleaved, i.e. each word output on the bus in any given clock cycle will contain consecutive data units from a single specific data stream; data from different data streams will not be mixed into the same word, and data units are not re-ordered within a given data word with respect to their order of presentation at the input.

**[0041]** The resulting time-division-multiplexed data stream, having the highly desirable properties of constant width, interleaved and predictability (i.e. the order in which words from different streams are presented on the output bus is fixed, regular and is known in advance) can be processed much more simply than the different physically separate data streams of arbitrary width at the inputs. For example, buffering the data for any number of data streams can be accomplished using a single physical memory block that is logically partitioned into separate buffer units utilizing an addressing arrangement which ensures that the sequence of data words are written to the appropriate buffer. Alternatively the output data can be placed on a single wider bus that is connected to some other processing entity, permitting the processing entity from having to be

replicated in order to deal with physically distinct data streams.

[0042] More particularly, the shuffle buffers which serve to accept data from upstream entities perform three principal functions, one of which is necessary if the shuffle buffers are used, and the other two being additional desirable features but are not absolutely required. These functions are described below, the first function being the necessary function.

[0043] 1. The input data are accumulated until sufficient data are available in each buffer. At this point, the data are read out in a shuffled order relative to the order in which they were written to the buffer. The shuffling is performed differently depending on the ratio of the width of the input data stream to the width of the output bus 13. The purpose of the shuffling is to properly order the data input to the pipelined butterfly network such that they may appear in interleaved fashion at its outputs. The shuffling is done in a deterministic manner, which is described in more detail later.

[0044] 2. In the event the data are arriving in an intermittent or bursty fashion (i.e. with long gaps between blocks of data), the shuffle buffers may be configured to accumulate data until complete blocks of data are available within the buffer prior to outputting the data to the permutation network. Once a complete block is available, the shuffle buffer should write out the entire block in sequence (per the shuffling process described in item 1 above) with no breaks. The size of each block should be normally equal to the width of the output data bus from the pipelined butterfly network. The purpose of this is to ensure that the data presented on the output of the stream

merging device has no gaps within individual words.

[0045] It should be noted that an ancillary function that can be implemented by the shuffle buffer units is to present dummy data to the permutation network when it is empty, or when insufficient data are present to form a complete block. Various other useful functions related to block formation may also be implemented by the shuffle buffers, as will be described later.

[0046] The block formation function by the shuffle buffers is optional; the stream merging process will continue to operate in its absence, but in this case the output data may have "holes" in the words.

[0047] 3. In the event the input data streams are synchronous to different clock signals (as is common when different data streams are being generated by separate physical layer devices), the shuffle buffers may be configured to synchronize the data to a common clock reference. This synchronization process may be done in a well known manner used to transport data between different clock domains.

[0048] This synchronization function is an optional function of the shuffle buffer.

[0049] The permutation network rearranges the spatial order of the inputs from the upstream data sources before they are presented to the pipelined butterfly network. This is done to permit any arbitrary arrangement of input data streams (i.e. to allow arbitrary assignment of logical streams or components of streams to the physical wires on which data are presented to the shuffle buffers). For example, a particular 32-bit stream may be configured such that its constituent 8-bit byte lanes are scattered over the various input data buses in some random order, possibly

intermixed with byte lanes belonging to other streams. The permutation network should in this case be configured to reorder the positions of the streams such that the byte lanes for the 32-bit stream are contiguous and located on a specific boundary. This will be described in more detail later.

**[0050]** The pipelined butterfly network provides the actual merging and width-extension of the various data streams. This network is topologically related to a butterfly graph, and is comprised of three sub-sections: (a) an input delay network, which imposes different fixed delays (in units of clock cycles) on the various input streams, (b) a pipelined butterfly network, which switches and sequences the streams in successive stages to merge and extend the data, and (c) an output delay network, which functions similarly to the input delay network, but serves to align the outgoing data properly, such that interleaved words are placed on the output of the complete apparatus.

**[0051]** It should be noted that the invention does not mandate that all three of the units (shuffle buffers, permutation network and pipelined butterfly network) must be present for all embodiments. One or more of the above (other than the pipelined butterfly network) may be omitted according to the following criteria:

**[0052]** 1. In the event the input data streams are all of the same width (e.g. all 8-bit data streams, all 32-bit data streams, etc.) then the shuffle buffer stage may be omitted. This is because no shuffling of data is required if the data are all of identical width. Of course, other auxiliary functions that may be implemented by the shuffle buffers, such as block accumulation and data synchronization may still be required. In this case, a set of simple First In

First Out (FIFO) buffers can accomplish such processing.

[0053] 2. If the input data streams are logically grouped and organized on appropriate boundaries with respect to the signal lines connected to the pipelined butterfly network, then the permutation network may be omitted. For example, if the input data consists of all 8-bit data streams, or all 32-bit data streams, then the streams are inherently organized properly and permutation (or shuffling) is not required. If the input comprises a mixture of, say eight 8-bit streams and two 32-bit streams presented on 128 bit signal lines, but the 32-bit streams are grouped logically and placed on the first 64 lines (i.e. on 32-bit boundaries), and the 8-bit data streams are placed on the next 64 lines, then the shuffle buffers are required to handle the differing data widths but no permutation network is needed to properly organize the streams spatially.

[0054] The most basic embodiment of the invention, therefore, comprises the pipelined Butterfly network 11. The shuffle buffers 7 are added if data of different widths must be handled. The permutation network 9 is included if data must be re-organized to bring logically related data streams together on contiguous signal lines.

[0055] As has been already noted, the apparatus has the desirable property that its constituent parts may be rearranged to perform a data splitting function rather than a data merging function. This is depicted in Figure 3.

[0056] As seen in Figure 3, a reversal and mirroring of the blocks in the merging system is used to realize a splitting system. The splitting apparatus accepts a time-division-multiplexed stream of constant-width input data words on a wide input bus 13A; the input stream must be regular and repeating in the same format as the output of a

similar merging network. The apparatus then follows a reverse procedure to take each input data word, which belongs to a different output stream of some arbitrary (and different) width, and serialize the data word on to the appropriate physical signal wires assigned to that output stream at the output of the apparatus.

**[0057]** The wide input bus 13A inputs the wide data stream to the pipelined butterfly network 11A, which is a mirror image of the network 11 used in the merge system described with respect to Figure 2. Similarly, the outputs of the pipelined butterfly network 11A are input to the inputs of permutation network 9A. The output signals of the permutation network 9A are input to shuffle buffers 7A, and the outputs of shuffle buffers 7A are applied to mixed width stream output buses 15.

**[0058]** Since the basic operation and construction of the splitting network of Figure 3 are identical to the merge apparatus of Figure 2 but with mirror symmetry, no further description of the splitting apparatus will be made as its structure and operation will be understood by a person skilled in the art who has read and understood the reverse direction embodiment.

**[0059]** The structure and operation of each block will now be described in detail.

**[0060]** Note that W represents below the lowest common divisor of the width of each of the (narrow) data streams that are merged to form the wide time-division--multiplexed output, and N represents the ratio of the width of the output stream to W. As a specific example, if 8-bit, 32-bit and 64-bit streams are being merged by the invention to create a single 128-bit time-division-multiplexed output stream, then W is 8 (the minimum stream size is 8 bits, and

this is also the common factor among all the input streams) and N is 16 (there are 16 such 8-bit streams that can be multiplexed into a 128-bit output). Various other parameters will be defined as required. The fundamental data unit in this case is an 8-bit byte.

Shuttle Buffer

**[0061]** With reference to Figure 4, each shuffle buffer 7 is comprised of three primary sub-sections: a RAM buffer memory 19, write logic containing a write address generation counter 21 and some control logic 23, and read logic containing a read address sequencer 25 and some control logic 27. Input interface 28 receives serial data at its input and outputs the serial data to inputs of the RAM buffer memory 19. Output interface 29 receives shuffled data from the memory 19 and provides it to an output bus.

**[0062]** The RAM buffer memory 19 stores input data at memory locations controlled by the write logic, and holds and accumulates the data input to it from input 28 until it can be read out in shuffled order as controlled by the read logic. The buffer memory 19 is B by W bits in size, where B is the number of data units (words) that can be held and W is the width of each data unit as supplied to the permutation network 9. Typically, B is expected to be some multiple of the number of data units N that comprise a single data word applied on the output bus of the pipelined butterfly network; thus if the butterfly network output is 128 bits wide and the input data units are in terms of 8-bit bytes, the buffer memory will be some multiple of sixteen 8-bit bytes in size. The shuffling process requires this multiple to be a minimum of 1, as shuffling cannot begin until an entire output word's worth of data are present in the buffer; normal values for the multiple range between 2

and 3 (implying a 32 x 8 or 48 x 8 RAM). The purpose of having more than N units of storage in the RAM is to permit fresh data to be written into the buffer while previously stored data are being read out in a shuffled fashion.

**[0063]** The write logic generates the address sequence required for writing data into the RAM buffer, and also implements the control functions needed to prevent data from being written into the buffer when no free space exists (i.e., normal FIFO write control functions). The address sequence is very simple, being an incrementing series of addresses starting at 0 and wrapping around after the end of the RAM buffer has been reached. The write logic is virtually identical to standard write addressing and control logic used in common FIFO queue structures.

**[0064]** The read logic generates the special sequence of addresses that causes the data to be read out of the buffer memory in shuffled fashion. This logic is also very similar to that of standard FIFO queue read control units, but with two exceptions. Firstly, the series of read addresses generated for successive words read out of the FIFO is not sequential, but instead forms an interleaved pattern. Secondly, the read logic does not permit reading to begin until there is sufficient data to form a complete sequence (i.e., enough to form a complete data word at the output of the butterfly network).

**[0065]** The table 1 below gives some examples of the time sequence in which data must be read out for various ratios between the output and input data word sizes for various streams. It is assumed that the width of the output data word is always 16 bytes (i.e., the data unit being a byte of 8 bits). Successive rows of table 1 give the successive addresses generated by the read logic.

Table 1

| # byte lanes per Input Word (Intrinsic Input Word Width) | 1 (8 bits) | 2 (16 bits) | 4 (32-bits) | 8 (64-bits) | 16 (128-bits) |
|---|---|---|---|---|---|
| Read Addr #0 | 0 | 0 | 0 | 0 | 0 |
| Read Addr #1 | 1 | 8 | 4 | 2 | 1 |
| Read Addr #2 | 2 | 1 | 8 | 4 | 2 |
| Read Addr #3 | 3 | 9 | 12 | 6 | 3 |
| Read Addr #4 | 4 | 2 | 1 | 8 | 4 |
| Read Addr #5 | 5 | 10 | 5 | 10 | 5 |
| Read Addr #6 | 6 | 3 | 9 | 12 | 6 |
| Read Addr #7 | 7 | 11 | 13 | 14 | 7 |
| Read Addr #8 | 8 | 4 | 2 | 1 | 8 |
| Read Addr #9 | 9 | 12 | 6 | 3 | 9 |
| Read Addr #10 | 10 | 5 | 10 | 5 | 10 |
| Read Addr #11 | 11 | 13 | 14 | 7 | 11 |
| Read Addr #12 | 12 | 6 | 3 | 9 | 12 |
| Read Addr #13 | 13 | 14 | 7 | 11 | 13 |
| Read Addr #14 | 14 | 7 | 11 | 13 | 14 |
| Read Addr #15 | 15 | 15 | 15 | 15 | 15 |

**[00066]** The general process for obtaining the sequence of addresses to use in order to properly shuffle the data read out of the buffer may be described as follows.

**[00067]** Let N represent the number of atomic data units in each output word (at the output of the butterfly network), and let k represent the number of atomic data units in each input word for a given stream. The quantity d should be computed as being the ratio of N divided by k. This quantity is referred to as the step distance. Now the method below should be followed:

**[00068]** 1. Start the read address sequence at zero (i.e., let the first read address be 0) and read out the first data word.

[00069] 2. Increment the read address by the step distance d.

[00070] 3. If the incremented read address is greater than N then subtract N from the result and add 1 to it.

[00071] 4. Read the next data unit at the current read address.

[00072] 5. Repeat steps 2, 3 and 4 until the read address becomes 15 (or, equivalently, sixteen words have been read out of the buffer) then stop.

[00073] Note that the address sequence described above assumes that the buffer size B is only N data units. If B is some multiple of N, the same method is used to derive the sequence, but the first read address generated by step 1 of the method is offset by an incrementing multiple of N prior to using it to access the buffer. The effect is to divide the buffer into blocks of N units, and to read the data within a given block according to the computed sequence, after which the next block is read, and so on.

[00074] As previously mentioned, two optional features may be included as part of the functions to be implemented by the shuffle buffer: synchronization and data accumulation.

[00075] With respect to synchronization, it should be noted that the shuffle buffer structure resembles a standard synchronizing FIFO queue (with the exception of the read logic, which generates a variable sequence of addresses rather than a simple incrementing sequence as in a standard synchronizing FIFO). Therefore, a standard means of clock synchronization and transport of data values across clock boundaries may be employed to allow the read and write ports of the shuffle buffer to use different clock references.

[00076] Data accumulation is required when either the

input (write) data rate is lower than the output (read) data rate, or when gaps exist in the write data stream. Well known means of handling gaps in the data stream, as usually implemented in a regular FIFO queue, are employed on the write side of the shuffle buffer. On the read side, however, there may be periods when the buffer is either completely empty or does not contain enough data to permit the reading process to start (i.e., there are less than N data units in it). The shuffle buffer in this case may be constructed so as to send exactly N "dummy" (invalid) data values to the permutation network whenever this situation is encountered, and to continue to send groups of N dummy values until the FIFO contains N or more data items. This ensures that the data stream between the shuffle buffer and the permutation network is delimited in units of N, and avoids "holes" within the output data words produced by the pipelined butterfly network.

**[00077]** As many shuffle buffers, each of width equal to one data unit W, are required as there are data units in the input streams. A total of N buffers is therefore needed (according to the notation already described). All of these buffers can operate independently with regard to the input (writing) of data, but must be synchronized to each other with respect to reading, i.e., the same clock is supplied to all buffers for reading, and data unit #0 is read out of all buffers within the same clock cycle. This ensures that the data presented to the permutation network will be aligned with regard to the different data streams, a necessary condition for merging data so as to obtain properly ordered words at the output of the pipelined butterfly network. If this condition is not satisfied (i.e., the read-out of data from different buffers is not aligned) then the pipelined

Butterfly network will maintain interleaving with regard to the separate streams (i.e., it will not merge data units from different streams into the same output word) but there may be 'holes" in the output words, and data may be misaligned within individual output words.

Permutation Network

**[00078]** The function of the permutation network is to allow any arbitrary (but non-conflicting) assignment of input signal lines to data streams, or to byte lanes within a given data stream. Given such an arbitrary assignment, the permutation network can be configured to re-order the spatial distribution of the data streams to allow the pipelined Butterfly network to function properly. The permutation network may be formed from any rearrangeable multistage network, i.e., a network where any arbitrary one-to-one mapping between the set of inputs and the set of outputs may be implemented without blocking between paths, and a subset of the paths may be altered without disturbing the remainder. One of the simplest rearrangeable networks is the Benes network, which is well known in the literature.

**[00079]** As an example, a Benes network capable of connecting 8 input buses to 8 output buses in any one-to-one order is depicted in Figure 5.

**[00080]** As shown in Figure 5, the Benes network is comprised of a set of elements (or nodes 33, indicated by the circles) interconnected by wires (or arcs). The width of each arc of the network is equal to the number of bits W in the basic data units presented on the input data streams to the apparatus (typically, 8 bits). Each of the nodes of the network can be separately configured to act as a 'pass-through" or a "crossover".

**[00081]** A node is formed from a pair of multiplexer units

as shown in Figure 6. Each of a pair of signal inputs Input 1 and Input 2 is connected to inputs A and B of the multiplexers 35 and 37; Inputs 1 and 2 are respectively connected to corresponding inputs A of the multiplexers and to corresponding inputs B of the other multiplexers. A select control input signal is applied to the S (select) inputs of both multiplexers.

**[00082]** When the select control input signal is set to a logical 0, Input 1 is connected to Output 1 of one multiplexer and Input 2 is connected to Output 2 of the other multiplexer (i.e. the node is configured to pass data straight through). When Select is a 1, then Input 1 is connected to Output 2 and Input 2 is connected to Output 1 (i.e. the node is set up in a crossed configuration).

**[00083]** With this multiplexer arrangement, any one-to-one mapping can be set up between the inputs and outputs. An example of an arbitrarily chosen mapping for the 8 x 8 Benes network being considered herein as an example, and the corresponding Select control input signals required to be presented to the nodes, is shown in the table 2, 3 and 4 below.

Table 2

Mapping

In->Out

| |
|---|
| I0->O6 |
| I1->O5 |
| I2->O4 |
| I3->O3 |
| I4->O2 |
| I5->O1 |
| I6->O0 |
| I7->O7 |

Table 3

| S 00 | S 01 | S 02 | S 03 | S 10 | S 11 | S 12 | S 13 | S 20 | S 21 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |

Table 4

| S 22 | S 23 | S 30 | S 31 | S 32 | S 33 | S 40 | S 41 | S 42 | S 43 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |

**[00084]** Any other desired mapping will have some other unique combination of Select signals S that establishes a set of paths from the inputs to the outputs to satisfy that mapping. It is preferred that these Select signals should be statically configured prior to operation of the apparatus in accordance with the distribution of input data streams on the actual input signal lines, so as to re-order the data streams in a regular fashion (i.e., byte lanes belonging to the same data stream should be adjacent to each other, in ascending order, and aligned to natural boundaries).

[00085] As an example of such a rearrangement, consider the case of four 8-bit streams $A_0$, $A_1$, $A_2$ and $A_3$; two 32-bit streams $\{B_{03}, B_{02}, B_{01}, B_{00}\}$ and $\{B_{13}, B_{12}, B_{11}, B_{10}\}$ (where "{x, y, z, w}" represents the concatenation of byte lanes x, y, z and w); and one 64-bit stream denoted as $\{C_{07}, C_{06}, C_{05}, C_{04}, C_{03}, C_{02}, C_{01}, C_{00}\}$. If these streams are input in a "jumbled" order from left to right as follows:

$\{C_{06}, A_0, B_{00}, B_{01}, B_{02}, B_{03}, B_{13}, A_1, B_{12}, C_{07}, C_{05}, C_{04}, C_{03}, C_{02}, C_{01}, C_{00}, B_{11}, A_2, B_{10}, A_3\}$,

then a 16 x 16 Benes network with 8-bit wide arcs may be used to re-order the streams into the regular form:

$\{C_{07}, C_{06}, C_{05}, C_{04}, C_{03}, C_{02}, C_{01}, C_{00}, B_{03}, B_{02}, B_{01}, B_{00}, B_{13}, B_{12}, B_{11}, B_{10}, A_3, A_2, A_1, A_0\}$

which is required by the pipelined butterfly network to operate properly.

[00086] As can be seen from the example, the byte lanes for individual streams must be grouped together in descending order, and the streams must be aligned on proper boundaries (64-bit streams on 64-bit boundaries, 32-bit streams on 32-bit boundaries, and 8-bit streams on any boundary).

[00087] Benes networks can be constructed for an arbitrarily large total number of input data units N in the input data streams. For a system having N input data units, where N must be a power of 2, the Benes network requires (2 x $\log_2 N$-1) columns of multiplexer nodes.

[00088] It will be understood by a person skilled in the art that register elements may be interposed between stages of the permutation network in order to pipeline the network and permit it to operate at high speeds. Further, the permutation network may be placed upstream of the shuffle buffers rather than downstream, so that the data is re-

arranged in the spatial domain before being re-ordered or shuffled in the time domain, rather than after.

[00089] As noted earlier, the permutation network may be omitted if the input data streams are already properly arranged.

Pipelined Butterfly Network

[00090] As mentioned earlier, the pipelined butterfly network performs the actual data width extension and merging functions. As shown in Figure 7, it is comprised of an input delay network 39, which feeds a butterfly network 41, which feeds an output delay network 43. As in the case of the Benes network, the pipelined butterfly network increases in size according to the total number of data units N in the input data streams. An example of a 4 x 4 pipelined butterfly network for handling 8-bit data will be described below, and is shown in Figure 7.

[00091] The pipelined butterfly network shown can extend and merge four data streams A, B, C and D, each of 8-bits in width, into a time-division-multiplexed sequence of 32-bit words 1,0,2,3 at the output bus.

[00092] The uncrosshatched circles 45 represent simple registers (delay stages), and the cross-hatched circles M0 - M7 represent registers 47 with 2:1 multiplexers 49 at their inputs. All of the registers are clocked at the same time (i.e. the entire network is synchronous). Each multiplexer has a single select control input S that is used to select one of the two inputs which connect to the outputs of a pair of registers in adjacent input delay network output paths, as shown. The select inputs S are modified on every clock cycle in a regular and repeating pattern, as will be described later.

[00093] Figures 8A - 8F illustrate the state of the

network in 6 successive clock cycles, whereby four 8-bit streams of data at the inputs may be organized by the shown 4 x 4 network into interleaved 32-bit data words at the outputs.

**[00094]** It is assumed that all of the input streams present their first byte, their second byte, their third byte, etc. in unison on successive clock cycles. After six clock cycles, the first 32-bit word of data (containing four consecutive bytes drawn from the first input stream) will appear at the output bus. From then on, successive 32-bit words of data belonging to consecutive streams will be placed on the output bus on every clock.

**[00095]** In Figures 8A - 8F, the input data streams are represented by {A1, A2, A3, A4, A5, A6…}, {B1, B2, B3, B4, B5, B6…}, {C1, C2, C3, C4, C5, C6…}, and {D1, D2, D3, D4, D5, D6…}; the output data words follow the sequence {A1, A2, A3, A4}, {B1, B2, B3, B4}, {C1, C2, C3, C4}, (D1, D2, D3, D4}, {A5, A6, A7, A8}, {B5, B6, B7, B8}, etc.

**[00096]** Turning to Figure 8A, it may be seen that the input data stream A(x) is received at input A, data stream B(x) is received at input B, data stream C(x) is received at input C and data stream D(x) is received at input D. Data stream A(x) is passed directly without delay to an input register 41A of the butterfly network 41; data stream B(x) is passed to the input register 41B of the butterfly network through one delay unit (e.g. clock period) 45A; data stream C(x) is passed to the input register 41C of the butterfly network through two delay units 45B; data stream D(x) is passed to the input register 41D of the butterfly network through three delay units 45C.

**[00097]** To generalize, each respective successive parallel input data stream passes through a delay network

which contains P serial delay stages, where P = (M-1), M being a whole number counting from 1 representing a count of each respective successive input data stream, for receiving the streams of data and passing each stream to an input of a corresponding multiplexer of a first stage of the multiplexers. It may be seen that the first data stream does not encounter any delay in the input delay network.

**[00098]** It should be noted that while the input register of the butterfly network is considered to be part of the butterfly register, it could as easily be considered as the last delay element of the delay network for each data stream. In that case the "-1" would disappear from the above equation. In this specification, the equations should be considered to be the same, depending on whether the input register is or is not considered to be part of the delay network.

**[00099]** The first byte of each of the data streams is stored, with the first clock pulse (clock #1), in the first delay (e.g. register) it encounters, as is shown in Figure 8A.

**[000100]** The outputs of each pair of the input registers of the butterfly network 41 are connected to each of the two inputs of a pair of multiplexers. Thus the output of register 41A is connected to the 0 inputs of multiplexers 42A and 42B; the output of register 41B is connected to the 1 inputs of multiplexers 42A and 42B. Similarly the output of input register 41C is connected to 0 inputs of multiplexers 42C and 42D, and the outputs of register 41D are connected to the 1 inputs of registers 42C and 42D.

**[000101]** The outputs of multiplexers 42A and 42B are connected respectively to the 0 inputs of multiplexers 42E and 42F, and are connected respectively to the 0 inputs of

multiplexers 42G and 42H. The outputs of multiplexers 42C and 42D are connected respectively to the 1 inputs of multiplexers 42G and 42H, and are also connected respectively to the 1 inputs of multiplexers 42E and 42F.

**[000102]** The general form of the array of multiplexers described is referred to herein as a butterfly network.

**[000103]** On the second clock pulse (clock #2), the data is shifted to the right, as is shown in Figure 8B. Thus the first byte A1 of data stream A(x) is shifted into the first multiplexer, and the second byte A2 is stored by the first register 41A. The first byte B1 of data stream B(x) is shifted from the delay (register) 45A into the input register 41B of the butterfly network, and the second byte B2 is stored in the delay (register) 45A. Similarly the successive bytes of the data streams C(x) and D(x) are shifted to successive delay (registers) as shown.

**[000104]** On the next clock pulse (clock #3), as shown in Figure 8C the byte A1 passes from multiplexer 42A to multiplexer 42E (see the select signal table below), and the byte B1 passes from the input register 41B to multiplexer 42A. Byte A2 passes from input register 41A to multiplexer 42B. The delays incurred by data streams C(x) and D(x) cause those data streams not yet to enter the multiplexer array.

**[000105]** The heavy lines in these figures represent the paths taken by valid data words within a given clock cycle; implicitly they represent the multiplexer select signals that must be used.

**[000106]** In Figure 8D, at the next clock pulse (clock #4), the first byte A1 of data stream A(x) has passed from multiplexer 42E to a first delay (register) 43A of the output delay network, while the second byte A2 is still in

the final multiplex stage, in multiplexer 42F. The first byte B1 of data stream B(x) follows byte A1 by one clock pulse, on the same line.

**[000107]** As shown in figure 8E, at the next clock pulse (clock #5) byte A1 is in the second delay (register) of the output delay network 43, and byte B1 is in the first. Byte C1 is in the last multiplexer stage, in multiplexer 42E, to follow the same delay path and bytes A1 and B1. Byte A2 is in the first delay (register) of the line that is adjacent to the one carrying byte A1. Bytes A3 and B2 are in the last multiplexer stage.

**[000108]** In Figure 8F, at the next clock pulse (clock #6), all of the bytes of each word of serial data stream A(x) have been transposed to a separate line, and appear in parallel at the same time on output bus 44. An examination of Figure 8F will also show that the bytes of serial data stream B(x) will follow by one clock pulse the parallel byte of data stream A(x). The parallel bytes of serial data stream C(x) and D(x) follow in succession, after which the process repeats for byte streams A(x), B(x), C(x) and D(x).

**[000109]** The serial data streams appearing on separate lines have thus been transposed into interleaved parallel time-division-multiplexed bytes and placed on an output bus.

**[000110]** With reference again to Figure 7, if a multiplexer select of '0' causes it to select the horizontal input, and a select of '1' causes the diagonal or cross input to be selected, then the following table 5 gives the multiplexer select signals required for the multiplexer nodes M0 - M7. Note that multiplexer nodes M0 - M7 shown in Figure 7 and referred to in the following table correspond to multiplexers 42A - 42H respectively.

Table 5

| Clock Cycle | M0 | M1 | M2 | M3 | M4 | M5 | M6 | M7 |
|---|---|---|---|---|---|---|---|---|
| Clock #1 | 0 | - | - | - | - | - | - | - |
| Clock #2 | 1 | 1 | - | - | 0 | - | - | - |
| Clock #3 | 0 | 0 | 0 | - | 0 | 0 | - | - |
| Clock #4 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | - |
| Clock #5 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| Clock #6 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| Clock #7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Clock #8 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 |
| . | . | . | . | . | . | . | . | . |
| . | . | . | . | . | . | . | . | . |
| . | . | . | . | . | . | . | . | . |

[000111] Following the table 5 above for control of the multiplexers, and the heavy lines on Figures 8A - 8F, it may be seen that the serial words in each input data stream are transposed to parallel bytes in an output data stream. As can be seen from the table, the multiplexer select signals are a simple and regular repeating pattern. The select signals for M0 - M3 (i.e. the first column of multiplexers) all toggle between 1 and 0 on every clock. The select signals (selects) for M4 - M7 toggle between 1 and 0 on every other clock. In general, for the $i^{th}$ stage of a pipelined butterfly network, (i=0 for the first stage) the multiplexer selects will toggle after $2^i$ clock cycles. In addition, the starting value for the sequence of toggles is offset by k modulo $2^{(i-1)}$ for the $k^{th}$ successive multiplexer in the $i^{th}$ stage of multiplexers, as can be seen from the table. The resulting control logic is thus exceedingly simple to implement, comprising a set of $2^{(i-1)}$ toggle flip-flops for the $i^{th}$ stage of multiplexers, with the toggle

flip-flops being loaded with a constant pattern on reset and then always toggling on the appropriate clock cycle thereafter.

**[000112]** A pipelined butterfly network may be built for any arbitrary number of input streams comprising N data units in total, where N must be a power of 2. To build such a network, there must be a total of (N x (N-1)/2) register units in the input delay network, N x $\log_2 N$ multiplexer units, N x ($\log_2 N$ + 1) register units in the actual butterfly network, and (N x (N-1)/2) additional register units in the output delay network. The total latency, in clock cycles, from the input of the first data unit on the first (topmost) stream to its emergence at the output is ($\log_2 N$ + N).

**[000113]** An example of an 8 x 8 pipelined butterfly network, capable of handling 8 streams of input data A - H and generating an 8-wide time-multiplexed sequence of output data 0 - 7, is shown in Figure 9. The reference numerals and letters used are similar to those of Figures 8A - 8F. Operation is similar to that of Figures 8A - 8F, and the person skilled in the art should make reference to that description for an understanding of operation of the embodiment of Figure 9.

**[000114]** The pipelined butterfly network has been described so far as handling data streams that are all of the same width, the width being equal to the smallest data unit (typically, this is 8-bits, although any arbitrary number of bits may be used without loss of generality). It is sometimes necessary to deal with data streams of different widths in the same apparatus (as, for example, when merging data streams from different physical layer devices having different speeds). It is possible to

accomplish this with the use of the shuffle buffers previously described, along with special control of the multiplexer stages in the pipelined butterfly network. The present invention can, as a result, deal with data streams whose widths are related to each other in powers of 2, and are also less than or equal to the width of the output data bus, and wherein ths sum of the input data stream widths does not exceed the width of the output data bus.

**[000115]** Two configuration parameters must be changed in order to deal with a data stream of width k units, where k is a power of 2. Firstly, the read ports of the shuffle buffers for all the byte lanes of the given data stream must all be configured to shuffle the data being read out, according to the method described earlier.

**[000116]** Secondly, the first $\log_2 k$ stages of multiplexers of the pipelined butterfly network must not be allowed to toggle, but must be frozen with their select inputs configured in straight-through mode. This is depicted pictorially in Figure 10, which considers 5 input streams (4 of which are 8-bits wide and 1 of which is 32-bits wide), that must be merged on to a 64-bit output bus using 8 shuffle buffers and an 8 x 8 pipelined butterfly network.

**[000117]** The permutation network is omitted in this drawing for simplicity, the data streams being assumed to be grouped and aligned on natural boundaries. The components of the 32-bit data stream are denoted as {A0, A1, A2, A3}, and the four 8-bit streams are B, C, D and E.

**[000118]** As may be seen from Figure 10, the first four byte lanes A0, A1, A2 and A3 belong to one 32-bit stream and are hence shuffled, while the next four byte lanes B, C, D and E are assigned to four independent 8-bit streams and are output unshuffled from the shuffle buffers. The pipelined

butterfly network is also differently configured from the previous embodiment; as denoted by the dashed lines, the first two stages (k=4 units for the 32-bit stream, and $\log_2 k=2$)of multiplexers for the first four byte lanes are frozen in a straight-through configuration, and all the rest of the multiplexers operate otherwise normally as described earlier.

**[000119]** The resulting apparatus produces a sequence of data on the eight output byte lanes as shown in the following table 6 (note that the clock cycles neglect the time taken to accumulate 8 bytes into the shuffle buffers).

Table 6

| Clock Cycle | Lane #0 | Lane #1 | Lane #2 | Lane #3 | Lane #4 | Lane #5 | Lane #6 | Lane #7 |
|---|---|---|---|---|---|---|---|---|
| 0 | - | - | - | - | - | - | - | - |
| . . . | . . . | . . . | . . . | . . . | . . . | . . . | . . . | . . . |
| 12 | A00 | A10 | A20 | A30 | A01 | A11 | A21 | A31 |
| 13 | A02 | A12 | A22 | A32 | A03 | A13 | A23 | A33 |
| 14 | A04 | A14 | A24 | A34 | A05 | A15 | A25 | A35 |
| 15 | A06 | A16 | A26 | A36 | A07 | A17 | A27 | A37 |
| 16 | B0 | B1 | B2 | B3 | B4 | B5 | B6 | B7 |
| 17 | C0 | C1 | C2 | C3 | C4 | C5 | C6 | C7 |
| 18 | D0 | D1 | D2 | D3 | D4 | D5 | D6 | D7 |
| 19 | E0 | E1 | E2 | E3 | E4 | E5 | E6 | E7 |
| . . . | . . . | . . . | . . . | . . . | . . . | . . . | . . . | . . . |
| | | | | | | | | |

**[000120]** In the above table 6, $A_{xy}$ denotes the $y^{th}$ byte input on byte lane x of the 32-bit data stream; $B_y$, $C_y$, $D_y$ and $E_y$ indicate the $y^{th}$ bytes of the 8-bit data steams, respectively. As can be seen, the output is extended to 64 bits, interleaved, aligned and time-multiplexed in the order of the byte lanes. Clearly the objectives of the invention

have been achieved.

[000121] A person understanding the above-described invention may now conceive of alternative designs, using the principles described herein. All such designs which fall within the scope of the claims appended hereto are considered to be part of the present invention.